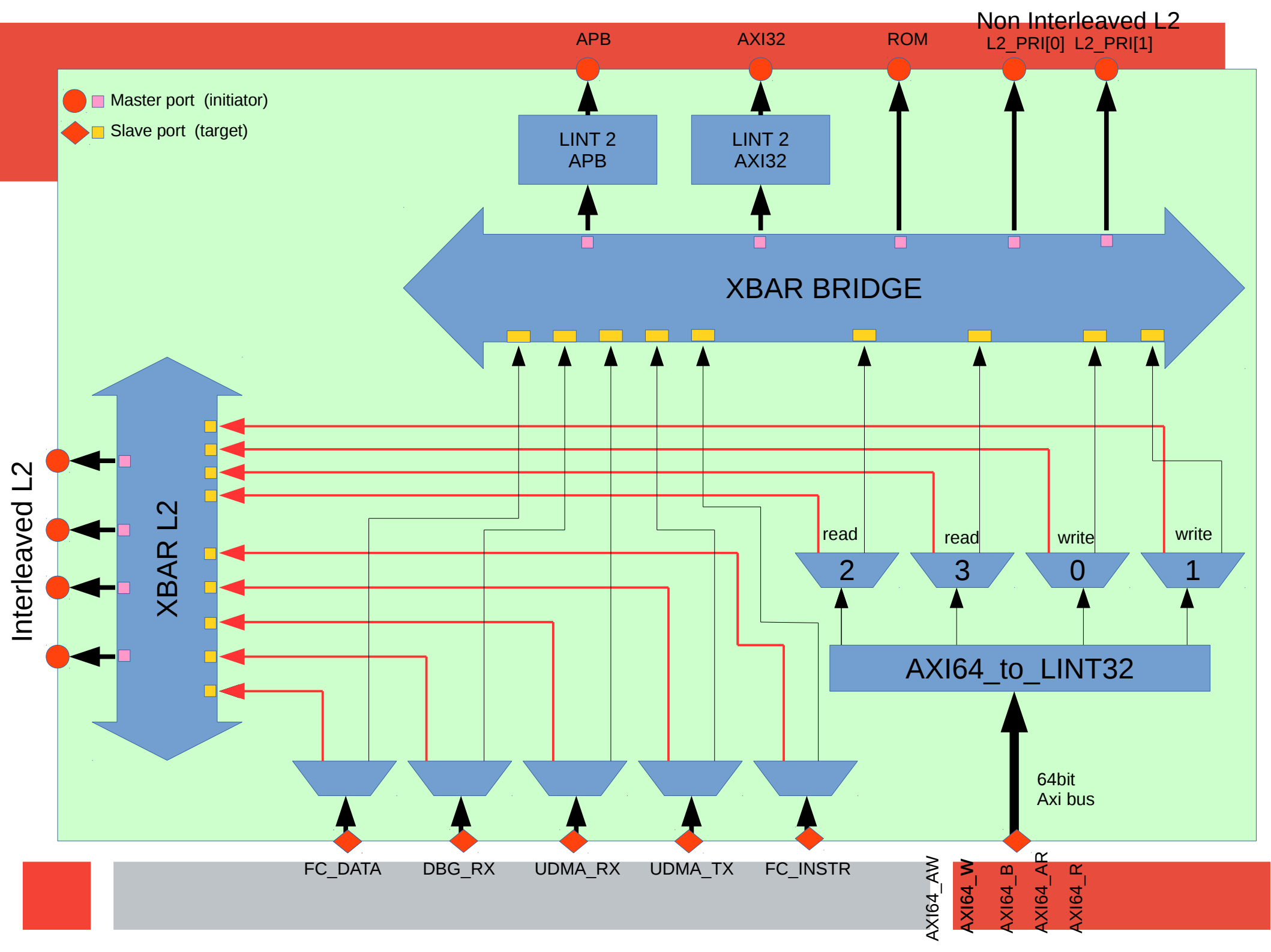
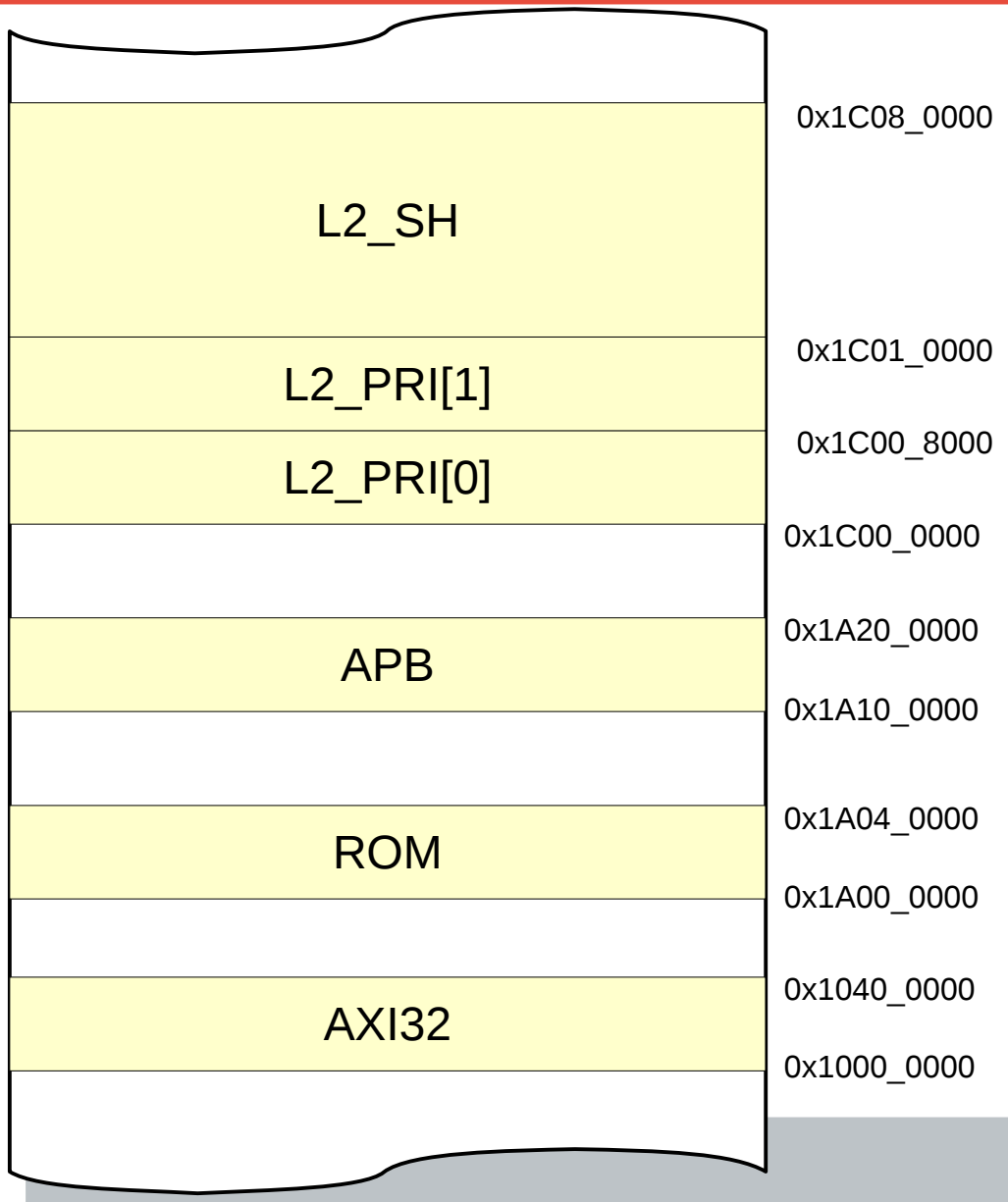
L2 Hybrid Interco


Non Interleaved L2
APB
AXI32
ROM
L2_PRI[0]
L2_PRI[1]
Master port (initiator)
Slave port (target)
LINT 2
APB
LINT 2
AXI32
XBAR BRIDGE
Interleaved L2
XBAR L2
read
read
write
write
2
3
0
1
AXI64_to_LINT32
64bit
Axi bus
FC_DATA
DBG_RX
UDMA_RX
UDMA_TX
FC_INSTR
AXI64_AW
AXI64_W
AXI64_B
AXI64_AR
AXI64_R

L2 Hybrid Interco
Memory space allocation



Address ranges are hardcoded in the fc_interconnect.

Decoding is made in two steps:

1) Demuxes: check if the transactions falls in the L2 shared

2) if not send the transaction to the bridge XBAR for further decoding

L2 Hybrid Interco Decoding: L2 shared

In the range of 0x1C01_0000 → 0x1C08_0000 transaction are sent to L2_shared. This region is processed in the XBAR_L2 which is a word interleaving xbar based interconnect.

Word level interleaving is made at granularity of 32 bit. 4 Memory channels are mapped on this interconnect, and these channels uses a log_intc protocol based on a request-grant flow control, with a response that comes the cycle after the request is granted.

Mapping:

Channel 0: --> 0x00, 0x10, 0x20, 0x30 etc

Channel 1: --> 0x04, 0x14, 0x24, 0x34 etc

Channel 2: --> 0x08, 0x18, 0x28, 0x38 etc

Channel 3: --> 0x0C, 0x1C, 0x2C, 0x3C etc

L2 Hybrid Interco Decoding: L2 shared

The L2 shared interface is a memory based interface, where no flow control is used:

```
output logic [N_L2_BANKS-1:0][DATA_WIDTH-1:0]          L2_D_o,
output logic [N_L2_BANKS-1:0][ADDR_L2_WIDTH-1:0]       L2_A_o,
output logic [N_L2_BANKS-1:0]                          L2_CEN_o,
output logic [N_L2_BANKS-1:0]                          L2_WEN_o,
output logic [N_L2_BANKS-1:0][BE_WIDTH-1:0]            L2_BE_o,
input  logic [N_L2_BANKS-1:0][DATA_WIDTH-1:0]          L2_Q_i,
```

The parameter **`N_L2_BANKS`** can be extended from 1 to any number power of 2

The related decoding addresses can be found in the fc_interco.sv

```
localparam logic [ADDR_WIDTH-1:0] TCDM_START_ADDR = {32'h1C01_0000}; // Start of L2 interleaved
localparam logic [ADDR_WIDTH-1:0] TCDM_END_ADDR   = {32'h1C08_0000}; // END of L2 interleaved
```

Be sure that the address range is coherent with L2 cut sizes and number of N_L2_Banks

L2 Hybrid Interco Decoding: L2 Private

To reduce the pressure on the L2, when cluster cores and DMA are using the Shared L2, this fabric interconnect provides additional 2 memory channels that are mapped on contiguous regions 1x1C00_0000 → 1x1C01_0000 on the XBAR_Bridge

This allows fabric controller or other agents to load/store from this region of the L2, without any congestion effects due contention on the interleaved region (L2 shared)

Decoding is performed in the xbar_bridge.

The parameter **`N_L2_BANKS_PRI`** can be extended, (also the addresses must be updated), but cannot be set to 0 without removing the related ports.

```
// CH_3, CH_4 Private Mem Banks (L2)
output logic [N_L2_BANKS_PRI-1:0][DATA_WIDTH-1:0]        L2_pri_D_o,
output logic [N_L2_BANKS_PRI-1:0][ADDR_L2_PRI_WIDTH-1:0] L2_pri_A_o,
output logic [N_L2_BANKS_PRI-1:0]                        L2_pri_CEN_o,
output logic [N_L2_BANKS_PRI-1:0]                        L2_pri_WEN_o,
output logic [N_L2_BANKS_PRI-1:0][BE_WIDTH-1:0]          L2_pri_BE_o,
input  logic [N_L2_BANKS_PRI-1:0][DATA_WIDTH-1:0]        L2_pri_Q_i
```

L2 Hybrid Interco Decoding: ROM

In case of ROM access (from 0x1A00_0000 to 0x1A00_4000) transaction are routed to the ROM port. This port talks native logarithmic interco protocol.

This port is read-only, thus the be-wdata-wen signals are not propagated outside the fc_interconnect module.

Signals used on this side are:

```
// CH_2 --> ROM
output logic                            rom_csn_o,
output logic [ROM_ADDR_WIDTH-1:0]       rom_add_o,
input  logic [DATA_WIDTH-1:0]           rom_rdata_i,
```

if you want to change the ROM SIZE, be sure that the **DATA_WIDTH** and **ROM_ADDR_WITH** are set properly.

L2 Hybrid Interco Decoding: APB

In case of APB access (from 0x1A00_0000 to 0x1A00_4000) transaction are routed to the APB bridge. The Bridge performs a protocol adpation between the log interco and the APB protocol. APB bridge is designed from scratch, beacause full timing isolation is required across the APB bridge. The request is sampled, and processed in the cycle after. When response comes back from the APB side, the read_data is sampled and delivered the cycle later.

FSM os this Bridge is quite simple and described in the **`lint_2_apb.sv`** verilog file.

The FSM basically is listenining for any lint request. As soon the requested is received from the LINT side, the FSM makes the APB request and waits for the READY signal form APB. Once is received it switch to its final stated to dispatch the DATA and comes back to the IDLE state.

L2 Hybrid Interco Decoding: AXI32

To allow the fabric controller to reach the cluster, the fc_interconnect is equipped with a 32bit master AXI port.

Since the transactions comes through a log interconnect, there is no way to generate AXI burst transactions, so only single-beat write or read operation are possible on this interface.

The LINT_2_AXI32 bridge is in charge to perform this protocol conversion. The main engine is made by a FSM with a blocking Behavior. Once the request reaches the Bridge, the related channels signaling are generated (AW-W, or AR), then the FSM listen for the competition of the pending operation (B or R valid signal).

A Size converter is needed to interface this port with 64Bit AXI interface!!

Demuxes

The First decoding stage is made on the demux and this decoding will identify the destination region: Bridge Side, Shared L2 side, or Decoding error (in case of access to unmapped region).

This block is receiving the address segment for each master port, and is calculating the one-hot destination vector.

MSB identifies the Shared L2 side

Bit 0 to bit MSB-1 identify the different master ports.

In case of request on unmapped region , the demux asserts an error message with r_ocp set to 1 and BAD_ACCES message in the read_data:

```
ERROR:
begin
    data_r_valid_o = 1'b1;
    data_r_aux_o   = sampled_data_aux;
    data_r_rdata_o = 32'hBAD_ACCE5;
    NS             = IDLE;
    data_r_opc_o   = 1'b1;
end
```

AXI64_to_LINT32

This is one of the most critical block of the design. The main purpose of this block is to convert an AXI64 transfer in a 32bit LINT transaction, without loosing bandwidth (eg from request from the cluster DMA or shared icache).

This conversion is made in 2 step:

1) AXI64 to LINT64

2) LINT64 to LINT32

Names are self explanatory. AXI64_to_LINT64 is a modified version of the AXI engine used in the L2 multi-bank, multi-port IP, and has been extended to support variable latency on the read_response (eg the APB side). This module is Blocking, and can process at full rate, if there is no congestion on the LINT side.

The LINT64_to_LINT32 perform a size conversion from 64 to 32 bit, using two channels on the LOG interconnect (both BRIDGE and L2 sides). 64 bit operations uses 2 channels in parallel (with consecutive addresses), while 32 bit axi transactions (SIZE field) uses only one channel (no unnecessary pressure on the xbars)

FLEXIBILITY

The IP is built using generic components, that can be easily reused in several other projects.

If you wish to make any change, you will need to make changes only at fc_interconnect level. Since the nature of this interco is highly heterogeneous, it has no sense to pack in a super-generic IP, hence it is a good idea to pack the sub-IPS in separated repositories.

For instance, you can change easily the number of shared ports from 4 to 8 or 2, you can remap the addresses, but if you wish to add-remove some ports, you will have to make some changes in the code to adapt to your needs.

The fc_interconnect is coming with additional signals aux, that can be used to support multiple outstanding transactions, or can be used to extend the functionality or to propagate side-band signals